doi:10.11918/j.issn.0367-6234.2015.05.005

一种新的过程间静态切片快速算法

苏小红, 龚丹丹, 王甜甜, 马培军

(哈尔滨工业大学 计算机科学与技术学院, 150001 哈尔滨)

摘 要: 针对传统的基于PDG、SDG的程序切片算法需要计算与程序切片无关的数据依赖而导致计算复杂度高的问题,提出一种新的过程间静态切片快速算法.该算法无需使用PDG、SDG的程序中间表示形式,而是根据TOKEN序列和复合语句控制结构信息表,将程序表示为idUCf五元结构,并在此基础上计算程序的过程间静态切片.实验结果表明,该算法在保证多层嵌套结构程序的静态切片完整性的前提下,充分考虑了函数调用信息,降低了时间与空间复杂度.本算法只计算与切片相关的数据依赖、控制依赖以及函数调用信息,计算复杂度低.

关键词: 系统依赖图;静态切片;TOKEN序列;控制依赖;数据依赖

中图分类号: TP311 **文献标志码**: A **文章编号**: 0367-6234(2015)05-0025-07

A new fast algorithm for interprocedural static slicing

SU Xiaohong, GONG Dandan, WANG Tiantian, MA Peijun

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China)

Abstract: The data dependences irrelevant to program slicing should be calculated in traditional static program slicing algorithms based on PDG and SDG. To deal with this problem, a new fast algorithm for interprocedural static slicing is proposed in this paper, in which the program is represented as idUCF 5-tuple structure according to TOKEN and the information about control-flow of compound statement, and the interprocedural static slicing is calculated without using of intermediate representation of program, such as PDG and SDG. Experimental results show that the proposed method takes the information about function call into full account, reduces time and space complexity and ensures the integrity of static slicing for program with multi-nested structure. The algorithm only calculates the data dependence, control dependence and function call relevant to slicing. Thus, it has a low computation complexity.

Keywords: system dependence graph; static slicing; TOKEN; control dependence; data dependence

程序切片技术是一种重要的程序分析和理解技术,广泛应用于程序调试、测试及软件维护中[1-2].其原理和方法最早出现在文献[3]中,文献[3]根据数据流方程的迭代解对程序切片进行了定义,并提出了基于控制流图(control flow graph, CFG)的计算过程内程序切片的算法,但此方法所消耗的时间和空间均比较多,计算所得的程序切片准确性也不是很好.文献[4]引入了基于程序依赖图(program dependence graph, PDG)的图形可达性算法,用于计算过程内切片,但此算法只能计算一个过程内的切片问题,含有多个过程的程序切片计算问题并未得到解决.文献[5]通过把PDG扩展为系统依赖图(system dependence graph, SDG),以计算过程间切片,根据系统依赖图,将函数间的切片问题转化为图的可达性问题[6],从而解决了包含多个过程程序的切片计算问题.

目前,计算程序静态切片主要使用的方法是基于依赖图的遍历算法[7].系统依赖图能够充分表示程序的数据依赖、控制依赖信息以及函数调用信息,但是SDG数据流分析的复杂度较高,而且生成系统依赖图的过程中,需要计算与切片无关的数据依赖,这些不必要的计算造成了时间和空间资源的浪费,并严重影响了切片计算的效率.

收稿日期: 2014-03-07.

基金项目: 国家自然科学基金(61173021, 61202092);上海市自然科学基金(15ZR1421400).

作者简介: 苏小红(1966—),女,教授,博士生导师;
马培军(1963—),男,教授,博士生导师.

通信作者: 龚丹丹,gongdandan0418@126.com.

类似地,传统的动态切片算法使用动态依赖图,复杂性也很高,针对该方法,文献[8]提出利用D/U表达式计算动态切片的方法,不需要使用动态依赖图,可以同时计算程序的控制依赖和数据依赖,空间开销较小.但传统的利用D/U表达式计算动态切片的方法,只考虑了直接控制依赖关系没有考虑多层嵌套的情况,而且由于未考虑函数调用的情况,只能用于过程内动态切片的计算中.

本文提出的基于idUCf五元结构的过程间切片算法,既保留了程序的数据依赖、控制依赖以及函数调用信息,又充分考虑了多层嵌套情况,而且不需要使用SDG,无需计算与切片无关的数据依赖、控制依赖以及不相关函数调用信息.

1 idUCf五元结构及相关定义

本文在分析静态切片与切片准则的基础上,给出了复合语句控制结构信息表以及idUCf五元结构的定义,作为本文切片算法的研究基础.

定义1(静态切片[9]) 一个程序P的静态切片是由P中的一些语句集合,它包含了所有可能影响兴趣变量的语句,考虑了程序中所有可能的执行路径.根据切片方向的不同,可分为前向切片和后向切片.前向切片是指所有受兴趣点变量的值影响的语句的集合;后向切片是指程序中所有能够影响兴趣点变量的值的语句的集合.本文所描述的是后向切片.

定义2(静态切片准则[10]) 静态切片准则是一个二元组$C=(n,V)$,其中:V为程序P中变量集合的一个子集;n为程序P中的某个兴趣点(对应于P中的一条语句).

定义3(复合语句控制结构信息表(CNT)) 复合语句控制结构信息表是记录程序P中的所有复合语句(选择、循环)的开始、结束位置等关键信息的集合,它能够精简的记录程序的所有控制依赖关系,以下简称CNT(compound node table).其结构定义如图1所示.

图1中,key为词法分析时所对应的TOKEN,name为该TOKEN对应的源程序中的单词.在本文所使用的编译器中,经词法分析后将for语句对应的TOKEN记为"CN_FOR",则key的值为"CN_FOR",name记录了源程序中的单词,则name的值为"for".对于多分支的选择结构,不仅应该记录选择结构的结束位置(endID)和结束行(endline),还应该记录各个分支内部的结束位置(ifendID)和结束行(ifendline),以便进行精确的路径分析.例如图2,多分支选择结构if(第13行),其内部分支结束位置ifendline=16,总分支结束位置endline=20;对于33行的else分支,其内部分支结束位置即为总分支的结束位置,即endline=ifendline=20.循环结构无需考虑多分支情况,因此for的endline与ifendline的值相同.词法分析过程中记录了每个TOKEN的具体位置,由于图2为实例代码片段,所包含的信息不完整,因此很难表示复合语句开始TOKEN位置、结束TOKEN位置以及内部结束TOKEN位置.因此,并未记录图2中的beginID、endID以及ifendID,在图中用"—"表示.

```
typedef struct      //记录复合语句起止位置
{
int id;             // 复合语句id号
CString name;       // 复合语句名
int key;            // 关键字
CString file;       //所属文件
int beginline;      //复合语句开始行
int endline;        //复合语句结束行
int beginID;        //复合语句开始TOKEN位置
int endID;          //复合语句结束TOKEN位置
int ifendID;        //复合语句内部结束TOKEN位置
int ifendline;      //复合语句内部结束行
} COMPNODE;
```

图1 复合语句控制结构信息表结构

```
D:\aa.c
11 for ( )
12 {
13  if ( )
14  {
15    sum=x+y ;
16  }
17  else
18  {
19    f=aver(a,b) ;
20  }
21 }
```

	1	2	3
name	for	if	else
key	CN_FOR	CN_IF	CN_ELSE
file	D:\aa.c	D:\aa.c	D:\aa.c
beginline	11	13	17
endline	21	20	20
beginID	—	—	—
endID	—	—	—
ifendID	—	—	—
ifendline	21	16	20

图2 复合语句控制结构信息表实例

定义4(idUCf五元结构) idUCf五元结构$<i,d,U,C,f>$是表示程序中变量定义、使用、控制依赖关系、函数调用关系的表达式,其中:i为语句的行号,d为语句i定义的变量;U为语句i使用变量的集合,C为语句i复合关系集合,存储该语句所在的复合语句所对应的CNT的id号,f为语句i调用函数信息.这样的表达式称为idUCf五元结构.本文主要讨论赋值语句结点(AssignNode)与函数调用结点(FunctionNode)的idUCf五元结构,图3中列出了图2(D:\aa.c)中第15行AssignNode和第19行FunctionNode的idUCf五元结构,其中Ø表示空集.

	i	d	U	C	f
AssignNode:	15	sum	x, y	1, 2	Ø
FunctionNode:	19	f	a, b	1, 3	aver

图3 idUCf五元结构实例

2　idUCf 五元结构的静态切片模型

本文建立了 idUCf 五元结构的过程间静态切片模型(如图 4 所示)，主要包括以下 3 个步骤：

1) 通过词法分析生成 TOKEN 序列，在 TOKEN 序列基础上进行代码标准化[11]，最终生成标准化 TOKEN 序列.

2) 在标准化 TOKEN 序列基础上，分别生成函数列表、复合语句控制结构信息表 CNT 和 idUCF 五元结构.

3) 应用基于 idUCf 五元结构的过程间静态切片算法，得到过程间静态切片.

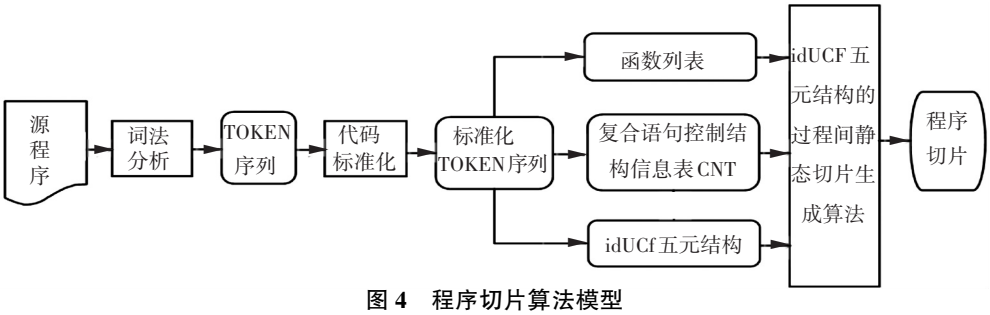

图 4　程序切片算法模型

3　切片算法

过程间静态切片，以切片准则 $C = (n, V)$，根据程序的 idUCf 五元结构 $< i, d, U, C, f >$，从源程序 P 中抽取结点 n 前，影响或间接影响变量 $v(v \in V)$ 的所有语句和函数.赋值语句以及函数调用语句直接影响变量的取值，本文将程序语句的操作分为赋值操作与函数调用操作，分别对应 idUCf 五元结构中的 AssignNode 与 FunctionNode. 当计算任意变量 v 的后向切片时，应逆序查找 idUCf 五元结构，直到找到 AssignNode 或 FunctionNode，使其满足 v = AssignNode 或 FunctionNode 的定义变量 d.因此，计算变量 v 的后向切片转化为计算相对应的赋值语句切片 AssignSlice (d) 或函数调用语句切片 FunctionSlice (d)，其中 d 为 AssignNode 或 FunctionNode 的定义变量为

$$\text{Slice}(v) = \text{AssignSlice}(d) \parallel \text{FunctionSlice}(d), \quad (v = d).$$

式中“ ‖ ”表示“或者”.

1) 当 d 为 AssignNode 的定义变量时，切片中应加入该赋值语句，并检查其复合关系集合 $< C >$ 及使用变量集合 $< U >$，首先，依次计算复合关系子切片(ControlSlice (c))，以保证多层嵌套程序的完整性. 当使用变量集合 $< U >$ 不为 $\emptyset$ 时，代表定义的变量 d 无法直接获得初始值，需要通过其使用的变量进行计算. 应依次递归计算其使用变量的子切片 slice $(u)(u \in U)$. 赋值语句结点切片 AssignSlice (d) 可确定为

$$\text{AssignSlice}(d) = \text{AssignNode}(d) + \sum_{u \in U} \text{Slice}(u) + \sum_{c \in C} \text{ControlSlice}(c).$$

式中“+”为所求切片的和.例如式中计算节点 AssignSlice(d) 由3部分的和组成，分别为该赋值节点 AssignNode(d)，使用变量的子切片 Slice$(u)(u \in U)$ 以及复合关系子切片(ControlSlice(c)).

当使用变量集合 $< U >$ 为 $\emptyset$ 时，代表定义变量被赋予常数，即找到了变量的初始值，此时，上式可变为

$$\text{AssignSlice}(d) = \text{AssignNode}(d) + \sum_{c \in C} \text{ControlSlice}(c).$$

当计算复合关系子切片 ControlSlice(c) 时，根据 idUCf 五元结构中的 c ($c \in C$; c 为该复合语句所对应的 CNT 的 id 号) 查找 CNT，找到所对应的复合语句的具体信息.

① 如果该复合语句为选择结构，则依次计算该选择结构的判断变量 j 的后向切片，以计算判断变量的初始值，即

$$\text{ControlSlice}(c) = \sum \text{Slice}(j).$$

② 如果该复合语句为选择结构，为了计算判断变量以及循环累加变量的初始值，首先应依次计算判断变量 j 和累加变量 a 的后向切片 Slice(j) 和 Slice(a)，为了保证静态切片的完整性，还应考虑 j 和 a 的值在循环体内是否改变，基于上述观点，在循环体内(beginline-endline)，顺序查找 idUCf 五元结构，当 j 或 a 与 AssignNode 或 FunctionNode 的定义变量 d 相同时，说明在循环体内，j 或 a 发生了改变，则应计算 AssignNode 或 FunctionNode 的定义变量 d 的后向切片.则有

$$\text{ControlSlice}(c) = \sum \text{Slice}(j) + \sum \text{Slice}(a) + \sum_{d=j \parallel d=a}^{\text{beginline-endline}} \text{Slice}(d).$$

2) 当 d 为 FunctionNode 的定义变量时，说明

此语句调用了函数,且 d 被赋予所调用函数的 return 返回值.切片中不仅应加入该赋值语句、复合关系子切片(ControlSlice),而且要在其调用函数中求得返回值切片 (Slice(r)),即

$$\begin{aligned}\text{FunctionSlice}(d) = {} & \text{FunctionNode}(d) + \\ & \sum_{c \in C} \text{ControlSlice}(c) + \\ & \text{Slice}(r) + \\ & \sum_{\text{if (para\_flag\_}i == 1)}^{m} \text{Slice}(\text{argue\_}i).\end{aligned}$$

式中:r 为所调用函数的 return 返回值;m 为被调函数的形参个数.在计算 Slice (r) 的过程中,还应记录每个形参的使用标志 para_flag.如果调用函数的第 i 个形参 (para_i) 为欲求 Slice(r) 的切片变量,并且 AssignNode(para_i) 的使用变量始终不为 Ø,则 para_flag_i 的值为 1,否则为 0.当 para_flag_i 的值为 1 时,说明被调函数中使用了主调函数的形参值,因此,应计算形参(para_i)所对应的实参切片 Slice(argue_i).

当所求切片变量 $d \in$ FunctionNode 的使用集 $< U >$,说明 d 为调用函数的参数之一.

① 当调用参数传递类型为传值调用时,被调函数中对形参的改变不影响实参的值,所以无需计算实参(para)所对应的形参切片 Slice(argue),则有

$$\begin{aligned}\text{FunctionSlice}(d) = {} & \text{FunctionNode}(d) + \\ & \sum_{c \in C} \text{ControlSlice}(c) + \\ & \text{Slice}(d).\end{aligned}$$

② 当所求切片变量 $d \in$ FunctionNode 的使用集 $< U >$,并且调用参数传递类型为传地址调用时,对形参的改变实际上是对实参的改变,因此,不仅要在调用函数中求得 d 所对应的形参切片 Slice(para_d),同时还应记录形参的使用标志 para_flag,当第 i 个形参的使用标志 para_flag_i 的取值为 1 时,说明被调函数的切片计算中需要使用第 i 个形参所对应的主调函数实参的取值,因此,应计算第 i 个实参的切片 Slice(argue_i).

$$\begin{aligned}\text{FunctionSlice}(d) = {} & \text{FunctionNode}(d) + \\ & \sum_{c \in C} \text{ControlSlice}(c) + \\ & \text{Slice}(\text{para\_}d) + \\ & \sum_{\text{if (para\_flag\_}i == 1)}^{m} \text{Slice}(\text{argue\_}i).\end{aligned}$$

本文在定义了上述公式的基础上,给出了基于 idUCf 五元结构的过程间静态切片算法,构造标准 $C = (n, V)$ 的程序切片 Slice(V) 算法如图 5 所示.主算法 Slice(V),根据切片准则 $C = (n, V)$,对第 n 行的每个变量 $d(d \in V)$,首先根据复合语句控制信息,判断其是否处于复合语句结构中,并调用 ControlSlice(c) 算法依次处理复合语句结构(解决了多层嵌套结构);然后,调用 Slicing(d) 算法,后向查找变量 d 的静态切片.

```
算法:Slice(V)
输入:源程序TOKEN序列
输出:程序切片
begin
  根据TOKEN序列计算本函数idUCf五元结构、复合语句控制结构信息表
while(V中还有未被处理的变量)
  读取变量d(d∈V)
    if(d处于复合语句结构中)
      ControlSlice(c)
        //对每个复合语句结构查找复合语句切片
  endif
  Slicing(d) //后向查找变量切片
endwhile
end
```

图 5 程序切片主算法 Slice(V)

复合关系子切片 ControlSlice(c) 算法如图 6 所示.算法 ControlSlice(c) 的功能是计算复合语句结构(选择、循环结构) 的程序切片,同时保持复合语句结构切片的完整性.对于复合语句结构中的变量 x,首先调用 Slicing(x) 算法查找变量 x 的切片,目的是为了找到 x 的初始值.然后在 beginline-endline 中查找 idUCf 的五元结构,如果 x 为第 k 个 idUCf 的定义变量,说明 x 在此复合结构中被重新赋值,则应依次处理第 k 个 idUCf 的使用变量集合 U.

后向查找切片变量 Slicing(d) 算法如图 7 所示.算法 Slicing(d) 的功能是根据 idUCf 五元结构计算变量 d 的后向静态切片.对于结点 d,后向查找 idUCf 五元结构.

① 如果 d 与第 i 个 idUCf 的定义变量相等,且 idUCf 类型为 AssignNode,则首先调用 ControlSlice(c) 函数,依次处理复合语句结构的程序切片,然后依次递归调用 Slicing(d) 函数处理变量使用集 U,直到 U 为 Ø.

② 如果 d 与第 i 个 idUCf 的定义变量相等,且 idUCf 类型为 FunctionNode,则调用 Slicing(r),在其调用函数中求得返回值切片并返回各形参标志 para_flag,根据 para_flag 的取值判断是否计算

形参切片.

```
算法:ControlSlice(c)
输入:TOKEN序列、控制结构ID、复合语句
      控制结构信息表、idUCf五元结构
输出:控制结构切片
begin
 if(复合关系未被处理过)
   加入切片;
 while(复合语句中还有未处理变量x)
 Slicing(x) //后向查找变量切片;
 查找beginline-endline对应的idUCf五元结构
   if(x为第k个idUCf五元结构的定义变量)
      加入切片;
      while(第k个idUCf控制依赖集C≠Ø)
      ControlSlice(c) //依次处理C;
    endwhile
    while(第k个idUCf使用变量集U≠ Ø)
          Slicing(u) //依次处理U中变量;
     endwhile
   endif
 endwhile
 endif
end
```

图 6　复合关系子切片 ControlSlice(*c*)算法

③如果 d 与第 i 个 idUCf 的使用变量集合中元素相等,且 idUCf 类型为 FunctionNode,根据实参-形参对应关系,调用 Slicing(para_d) 函数,在调用函数中求得形参切片并返回各形参标志,根据 para_flag 的取值判断是否计算形参切片.

4　实验结果和分析

根据本文算法,以切片准则(24,aboveAver),计算程序图 8 中的实例程序的静态切片.本文切片算法只计算切片点所在函数以及与切片点计算相关的调用函数时才计算 idUCf 五元结构和 CNT,切片计算结果为:切片 slice(24, aboveAver) = {24,4,20,40,48,45,43,42,41,19, 2, 10,9,8},切片集合中元素的顺序为其被加入的顺序.为此,本文使用行号来代表程序切片语句.因为行号与语句具有一一对应关系.图 8 中的阴影部分为本切片算法所分析的语句,由此可以看出,本文算法只对 main 函数以及 GetAboveAver 函数进行分析,只计算 main 函数以及 GetAboveAver 函数的 idUCf 五元结构和 CNT,无需计算与切片计算无关的 GetDetail 函数.

```
算法:Slicing(d)
输入:TOKEN序列、函数列表、idUCf五元结构、复合语句控制结构
      信息表
输出:变量后向子切片
begin
  从当前行后向查找idUCf
  if(找到AssignNode(i):第i个idUCf)
     加入切片
     while(AssignNode(i)控制依赖集C≠Ø)
          ControlSlice(c) 依次处理C;
    endwhile
    if((AssignNode(i)变量使用集U=Ø)
     if((AssignNode(i)不在选择分支中)||(AssignNode(i)在选择分支中
     &&每个分支都存在AssignNode(k)的定义变量== AssignNode(i)
     的定义变量&& AssignNode(k)的使用集合U=Ø))
        结束;
      else
        Slicing(d) //继续后向查找变量d的子切片
      endif
     else
       while(第i个idUCf使用变量集U≠Ø)
            Slicing(u) //依次处理U;
       endwhile
     endif
   if(变量d为FunctionNode(i)的定义变量)
         Slicing(r);// 在其调用函数中求得返回值切片并返回各形
                    参标志para_flag
      while(形参标志para_flag=1)
           Slicing(para);
    endwhile
  elseif(变量d∈FunctionNode(i)的变量使用结合U)
       if(调用参数传递类型为传值调用时)
          Slicing(d);
       elseif(调用参数传递类型为传地址调用时)
          Slicing(para_d)// 在调用函数中求得形参切片并返回各形
                          参标志
        while(形参标志para_flag=1)
            Sicing(para);
    endwhile
  endif
endifendif
end
```

图 7　后向查找切片变量 Slicing (*d*)

表 1~4 列出了 main 函数和 GetAboveAver 函数的 CNT 以及 idUCf 五元结构.

表 1　main 函数的 idUCf 五元结构表

i	d	U	C	f
6	failnum	Ø	Ø	Ø
7	sum	Ø	Ø	Ø
8	score	Ø	Ø	Ø
11	sum	sum,score	1	Ø
15	failnum	failnum	1,2	Ø
18	aver	sum	Ø	Ø
19	aboveAver	aver,score	Ø	GetAboveAver
20	Ø	score	Ø	GetDetail

表 2　main 函数的 CNT

id	name	beginline	endline	…
1	for	9	17	…
2	if	12	16	…

```
/*函数功能:1.统计不及格人数并打印其分数;2.计算平均分,统计成绩在平均分之上的学生人数并打印其分数;3.统计跟分数段学生人数及所占百分比.*/
1 # include<stdio.h>;
2 int GetAboveAver(float aver, float score[], int n);
3 void GetDetail(float score[], int n);
4 main()
5{
6   int failnum=0, aboveAver;
7   float aver,
8   float sum=0;
9   float score[10]={61,57,88,91,76,55,77,63,92,58};
10  for ( int i=0; i<10; i++ )
11  {
12   sum+=score[i];
13   if ( ( score[i] < 60 ) )
14   {
15     printf("The fail score is :%.0f\n",score[i]);
16     failnum++;
17   }
18  }
19  aver = sum / 10;
20  aboveAver = GetAboveAver(aver, score, 10);
21  GetDetail(score,10);
22  printf("The aver score= %.0f\n",aver);
23  printf("Fail students num= %d\n",failnum);
24  printf("Above aver num= %d\n", aboveAver);
25 }
26 void GetDetail(float score[], int n)
27 { int  i, j, stu[6];
28  for (i=0; i<6; i++)  stu[i]=0;
29  for (i=0; i<n; i++)
30  {   if (score[i] < 60)   j= 0;
31      else   j = ((int)score[i] - 50) / 10;
32      stu[j]++;
33   }
34  for (i=0; i<6; i++)
35  { if (i = 0)  printf("<60%d%.2f%%\n",
                        stu[i],(float)stu[i]/(float)n*100);
36   else if (i == 5)  printf("%d%d%.2f%%\n",(i+5)*10,
                          stu[i],(float)stu[i]/(float)n*100);
37   else  printf("%d--%d%d%.2f%%\n",(i+5)*10,
                 (i+5)*10+9,stu[i],(float)stu[i]/(float)n*100);
38  }
39 }
*****************************************************
40 int GetAboveAver(float aver, float score[], int n)
41{  int count=0 ,i ;
42   for (i=0; i<n; i++)
43   {  if (score[i] >= aver)
44     {  printf("Above aver score:%.0f\n", score[i]);
45        count++;
46     }
47   }
48   return count;
49 }
```

图 8　实例程序

表 3　GetAboveAver 函数的 idUCf 五元结构

i	d	U	C	f
40	count	Ø	Ø	Ø
44	count	count	1,2	Ø

表 4　GetAboveAver 函数的 CNT

id	name	beginline	endline	…
1	for	41	46	…
2	if	42	45	…

图 9 为本文算法所得到的静态切片程序,本文算法充分考虑了程序的控制依赖、数据依赖以及函数调用信息.文献[8]只考虑了直接控制依赖关系,而本文将直接控制依赖关系扩展为复合关系集合 $< C >$,充分考虑了多层嵌套情况,所得切片精度更高;而且,文献[8]中只计算了过程内动态切片,而本文算法可以计算过程间静态切片.

```
1 main()
2 {
3   float sum=0;
4   float score[10]={61,57,88,91,76,55,77,63,92,58};
5   for ( int i = 0; i< 10 ;  i ++ )
6   {
7      sum+=score[i];
8   }
9    aver = sum / 10;
10   aboveAver = GetAboveAver(aver, score, 10);
11   printf("Above aver num= %d\n", aboveAver);
12 }
*****************************************************
13 int GetAboveAver(float aver, float score[], int  n)
14 {  int count=0 ;
15   for (int  i=0;  i< n;  i++)
16    { if (score[ i] >= aver)
17        count++;
18    }
19   return count;
20 }
```

图 9　program1 的切片程序

传统的基于 PDG、SDG 的静态程序切片算法复杂度很高[12-13].例如对于程序 program1 (见图 9),传统方法生成的 SDG 共有 66 个结点(本程序为 37 行:除去“{”和“}”),66 条控制依赖边,79 条数据依赖边,2 条函数调用边.采用本文方法建立的 idUCF 五元结构的元素总个数为 10 个,CNT 中的元素总个数仅为 4 个,大大低于 SDG 中的节点个数.在计算程序切片过程中,传统的基于 SDG 的方法需要根据控制依赖边和数据依赖边遍历所有节点,而本文算法只需计算与切

片点相关的控制依赖、数据依赖以及相关的函数.在 main 函数中,控制依赖信息只分析了与切片点相关的第 10 行的 for 循环,无需分析第 13 行的 if 语句.数据依赖信息只分析了与 aboveAver 相关的 aver、score、sum 的数据依赖,不需对 failnum 的数据依赖进行分析.函数调用方面,函数 GetDetail 与切片点无关,也不用分析.另外,只计算与切片点相关函数的 idUCf 五元结构和 CNT,在本实例中,只计算了 main 函数和 GetAboveAver 的 idUCf 和 CNT,而 GetDetail 未计算.综上所述,本文算法在保证多层嵌套结构程序的静态切片完整性的前提下,充分考虑了函数调用信息,有效节省了时间与空间.

1)传统方法分析的对象是 PDG 或 SDG.建立 PDG 和 SDG 时需要计算程序的控制流和数据流,复杂度主要集中在数据流分析中. 假设程序中包含 n 个节点,利用迭代算法生成每个节点的定值/引用到达信息, 数据流分析的时间复杂度为 $O(n^2)$.

本文方法分析的对象是函数的 CNT 和 idUCF 五元结构.建立 CNT 和 idUCF 五元结构只需分析标准化 TOKEN 序列一次即可,因此时间复杂度为 $O(n)$.

2)传统方法计算切片时的时间复杂度普遍为 $O(n^4)$. 对于本文方法,考虑最坏情况,假设 idUCF 五元结构中元素个数为 n (实际上 idUCF 中的元素个数远小于 n), 切片准则中包含了程序中的所有变量,本文方法计算切片的时间复杂度为 $O(n^2)$.

对于空间复杂度,传统方法通常使用矩阵来存储 PDG、SDG 的数据依赖关系和控制依赖关系,矩阵大小为 $n \times n$, 则空间复杂度为 $O(n^2)$. 本文方法只需要存储 CNT 和 idUCF 五元结构.考虑最坏情况,假设 idUCF 五元结构和 CNT 中元素个数之和为 n (实际上应远小于 n), 本文方法需要存储空间为 $5 \times n$,则空间复杂度为 $O(n)$.

5　结　语

针对传统的基于 PDG、SDG 的程序切片算法需要计算无关数据依赖,计算复杂度较高的问题,本文提出基于 idUCf 五元结构的过程间切片算法,对于多层嵌套结构程序在保证其静态切片完整性的前提下,只计算与切片相关的依赖信息,提高了程序切片的计算效率.

参考文献

[1] WARD M, ZEDAN H. Combining dynamic and static slicing for analysing assembler [J]. Science of Computer Programming,2010, 75(3): 134-175.

[2] HALDER R, CORTESI A. Abstract program slicing on dependence condition graphs [J]. Science of Computer Programming, 2013, 78(9):1240-1263.

[3] WEISER M D. Rrogram slices formal, psychological and practical investigations of an automatic program abstraction method [D]. Michigan: University of Michigan, 1979.

[4] OTTENSTAN K J, OTTENSTAN L M. The program dependence graph in a software development environment [J]. ACM SIGPLAN Notices, 1984, 19(5):177-184.

[5] HORWITZ S, REPS T, BNKLEY D. Interprocedural slicing using dependence graths [J]. ACM Trans on Programming Language and System, 1990,2(1):26-60.

[6] 姜淑娟,徐宝文,史亮. 一种基于异常传播分析的数据流分析方法 [J]. 软件学报,2007,18(4): 832-841.

[7] 张龙杰,谢晓方,袁胜智.一种改进的静态程序切片算法[J]. 计算机应用, 2009,29(3): 705-711.

[8] BESZEDES A, GERGELY T, SZABO Z M, et al. Dynamic slicing method for maintenance of large C programs [C]//Proceedings of the Fifth European Conference on Software Maintenance and Reengineering. Washington, DC: IEEE Computer Society, 2001:105-113.

[9] SIKKA P, KAUR K. Program slicing techniques and their need in aspect oriented programming [J]. International Journal of Computer Applications, 2013, 70(3): 11-14.

[10] DANICIC S, HARMAN M, HOWROYD J, et al. A non-standard semantics for program slicing and dependence analysis [J]. The Journal of Logic and Algebraic Programming, 2007, (72): 191-206.

[11] 龚丹丹,王甜甜,苏小红,等.冗余代码缺陷检测方法[J].哈尔滨工业大学学报,2012, 44(7): 58-63.

[12] BINKLEY D, GOLD N, HARMAN H. An empirical study of static program slice size [J]. ACM Transactions on Software Engineering and Methodology, 2007, 16(2): 1-32.

[13] KRINKE J. Effects of context on program slicing[J]. Journal of Systems and Software, 2006, 79(9): 1249-1260.

(编辑　张　红)